

Code Review Checklist

An Ideal Code Review Checklist that applies for most programming languages

Purpose of Code Review

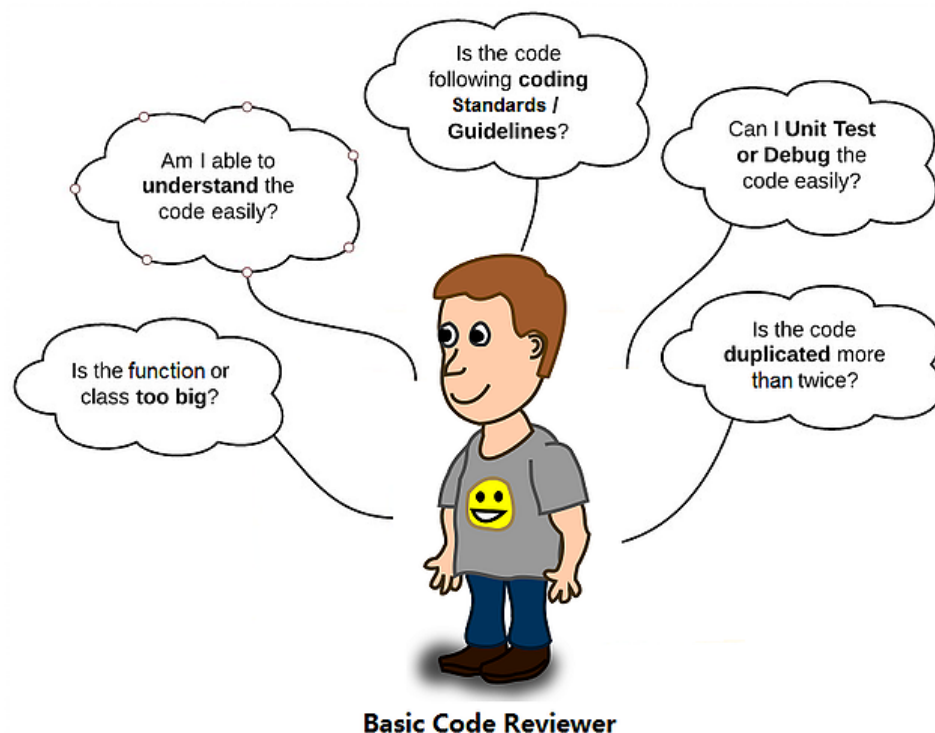
The ultimate purpose of code review is to investigate the code to find weak spots, faults, strengths and ways to optimize the code for better performance. It is mainly to deliver a bug-free (at least near perfect) application that meets the purpose (requirements) while meeting the industry standards.

The code review checklists are illustrated in two parts:

Code Review Checklist - Fundamental

Code Review Checklist - Comprehensive

Code Review Checklist - Fundamental



1) Objective based [Purposeful]

The code achieves its purpose. In simple terms, it does what it is supposed to.

2) Unbreakable [Validated]

Validations are used wherever necessary. The code never breaks under any circumstances. Especially under invalid inputs that come from the user end. Regardless of it being a negative, over-sized, invalid format, etc., every input passed should be processed, sanitized before taking it further. Every object is checked for its actual data existence before accessing its properties.

3) Responses are handled [Error handling and Data formatting]

Not just the error messages, every response that is returned by the server must be properly handled. It should have necessary headers, response messages, error codes and any other necessary details attached with it in required format. All possible scenarios are tested to avoid deadlocks, timeouts, etc.

4) Follows architecture [No design deviation]

Verify that the approved architecture/design is followed throughout the application (If there is none, consider putting it in place). If there are any design changes required, ensure that these are documented, baselined and approved before implementing them in the existing code.

5) Unit tested [Reliable]

Every core method has a unit test which passes.

6) Reusable [No repetition]

All methods serve a limited and clear purpose (follows DRY principle). Functions are reused wherever applicable and written in such a way that they can be re-used in the future implementations. There is no duplication of code. Logics make use of general functions without ambiguity.

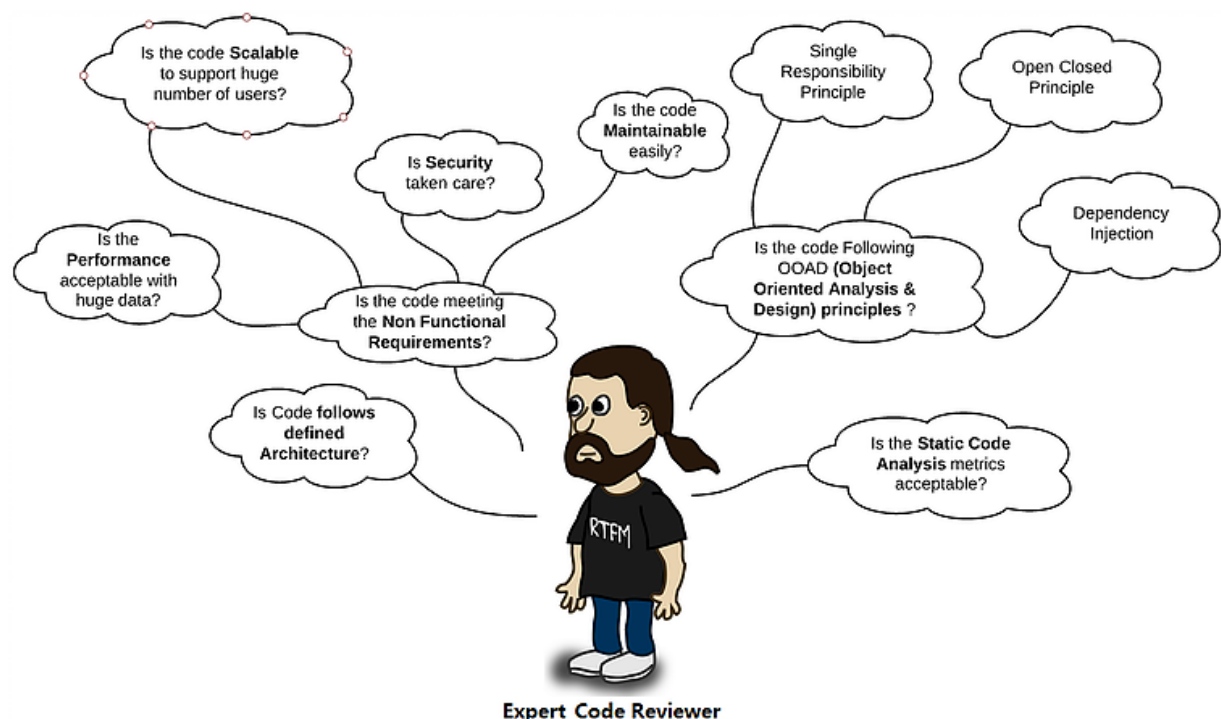
7) Performance oriented [Speedy response & Scalability]

The landing of the application is swift. There are no long delays between the requests and responses. Raw string concatenations are avoided and proper methods such as StringBuilder are used. The code is scalable and able to handle a large amount of data and upcoming features.

8) Secure [Safe code]

The code is secure in terms of authentications (with encryption), injections, roles, unauthorized access, directory browsing, SQL injection, cross-side scripting, etc. It follows the OWASP 10 security principles.

Code Review Checklist - Comprehensive



9) Manageable [Crisp and Formatted]

The code is readable, commented and easy to manage. It is friendly formatted and easy to read/understand. Methods are not too big to manage and they don't exceed readable size.

10) Meets coding conventions and standards [Standardized approach]

The code follows the coding conventions, standards and is consistent with the existing application code. There are no commented code and hard coded values.

11) Detach connections after usage [Memory handling]

Resources that are not automatically released after usage are freed. Connections, ports are closed properly.

12) Has configurable logging[Traceability]

Log every transaction or the ones that require logging. They are stored in a repository (as a file) as well as in the database (as text). Logging in different stages for different purposes can be enabled/disabled in the configuration file (like web.config)

13) Code coverage is more than 95% [Efficiency]

Code coverage is as important as the unit test cases passing. 95% of the code is covered (which means 95% code is actually tested via unit test cases).

14) On-demand resource delivery[Fast]

Resources are fetched and delivered only on demand. Necessary options are available for dealing with huge data such as paginations, etc.

15) No warnings / console logs [Data safety]

No compiler warnings arise while running the application. Logs that are used while developing are cleared and none of the application information (especially the sensitive ones) are written in the browser console.

16) Legal usage of third-party tools/libraries[Licencing]

External libraries are used only if proven necessary for the application. If there are third-party tools or libraries used, then the licenses and legal usages are verified and complaint.

How to prepare for effective code reviews: catch more bugs with these best practices

Code reviews. Two words that can elicit visions of the last hurdle to conquer before one's hard work is released into production.

For the reviewee, it can also be difficult having work scrutinized, line by line. For the reviewer, it can become a chore lacking direction and process.

However, code reviews don't have to be something that fills anyone with dread. Here are the three tips for more delightful and effective code reviews.



I Am Devloper

@iamdeveloper

Follow



10 lines of code = 10 issues.

500 lines of code = "looks fine."

Code reviews.

1:58 AM - 5 Nov 2013

8,035 Retweets 4,401 Likes



106 8.0K 4.4K

Ah code reviews!

1. Prepare code reviews

Preparing for the review is the responsibility of the reviewee. Two techniques which ensure more productive reviews are having small pull requests and providing information to the reviewers.

Small pull requests of 200 lines of code or less are the optimal size for catching defects in code reviews. This can be achieved by creating a pull request for one piece of functionality, unit tests, or, for large code changes, only submitting 200 lines of code at a time if possible.

Background information on the pull request goes a long way in giving reviewers context. It could include a description of the feature or bug fix, comments where necessary, and which solution, project or part of the system this code is found in.

2. Choose wisely

In Tekken Tag, you pick two characters you're strongest with to get the most out of the tournament. Choosing the right code reviewers is similar in that you work as a team to get the best out of the review while learning and enforcing a good work culture.

A good place to start is to have a clear purpose of the review, and then to pick the right team.

Purpose

Most code reviews cover one or a mix of the following four goals:

- Reinforce clean code practices
- Catch bugs
- Knowledge share
- Upskill inexperienced developers

Once you have established the goal, the reviewee or team assign the reviewers. Newer members of the team and less experienced developers may need direction on who the reviewer should be.

Reviewers

	Clean code	Catch bugs	Knowledge share	Upskill
TEAM			✓	✓
SENIOR	✓	✓	✓	✓
SENIOR + DEV	✓	✓		

code review matrix

There are three main types of reviewers. Depending on the above purpose, below is a clear breakdown of the attributes of each type of reviewee and how they relate to the purpose of the review:

Team

Attributes: A small feature team of two to four developers of mixed skills and experience.

If a team needs to share knowledge or upskill, getting the whole team involved is an excellent idea. Each member contributes by asking questions, catching typos and bugs, finding better ways to write code and upkeep in-company style rules.

Being able to ask questions during reviews and having healthy debates on the 'best way' lifts the overall skill level of a team. I worked on a small feature team where the whole team jumped on code reviews whenever code was ready. Reviewing code was the #1 way to learn the code base and view clean code techniques in action regularly.

Senior developer

Attributes: 'Senior' is a loose term here. This developer should have a high standard of clean code practices, excellent troubleshooting, and bug catching skills, and possibly hold a mentoring role.

A senior developer is suitable to review all four types of code reviews. The final step before shipping code usually lies at the code review. And they take time. A more experienced developer may be faster at spotting clean code violations and bugs. The faster this step, the faster it will reach the 'Done' column in JIRA.

Senior and second developer

Attributes: Senior with attributes stated above. The other developer, of any skill level, should know the area of code under review well, and possibly be more knowledgeable of the area than the senior dev.

In this tag team, time is cut down by sharing tasks. The senior dev could look at clean code techniques and catch bugs. The second developer could primarily focus on the functionality. There will be overlaps with both developers looking at code techniques and bug catching, but this will help catch most of the issues with two pairs of eyes.

Reviews are a practical way to reinforce a good company culture. Feeling like you're under scrutiny is never pleasant, but reviews don't have to feel that way


if there's a healthy company culture. Reviews are an excellent way to get one's code into tip-top shape – sarcasm and derision have no place in a code review.

3. Have a checklist

Checklists are a great way to include all elements of a code review, without having to remember what to look out for. Every company is different, but here's a checklist that covers the basics:

CODE REVIEW CHECKLIST

- ☐ Purpose established
- ☐ Background info provided
- ☐ Clean code practices
- ☐ Company code styles upheld
- ☐ Be kind
- ☐ Unit tests included where appropriate
- ☐ 20 minutes at a time



The checkbox '20 minutes at a time' is to help reviewers relax their eyes and come back to the code with a fresh perspective. What better way to catch those pesky typos or bugs!

Every team is different

Maybe these tipshelp us at GriCreative to provide proactive support for team members while ensuring code quality.

A sample of code review checklist is shown below. This is from the book 'Code Craft'.

Code review

CHECKLIST

Use this form to help you perform a code review.

About the code

Module name: _____
Version reviewed: _____
Code author: _____

Reviewed by: _____
Date: _____
Language: _____
Number of files: _____

Automated inspection

- ☐ The code compiles without errors
- ☐ The code compiles without warnings
- ☐ There are unit tests
 - ☐ They are sufficient (include all boundary cases, etc.)
 - ☐ The code passes them

- ☐ The code is kept under source control
- ☐ The code has been tested with inspection tools

Tool name	Results
_____	_____
_____	_____
_____	_____

☐ Continue to next section ☐ Stop review here

Design

- ☐ The code is complete (against its specification)
- ☐ There is a good choice of algorithms
- ☐ Optimizations are necessary and appropriate
- ☐ Any missing functionality is marked clearly in the code

General observations about the code's design

- ☐ The code is well structured
- ☐ There is design documentation
 - ☐ The code matches the documentation

☐ Continue to next section ☐ Stop review here

General code comments

Style

- ☐ The code layout is clear
- ☐ It follows project style guidelines
- ☐ There is a good (unambiguous) public API
- ☐ There is a good choice of names

Defensive programming

- ☐ Array access is guarded and safe (C/C++)
- ☐ There is a correct choice of types
- ☐ All input is validated
- ☐ There is no use of compiler-specific features

General comments

General comments about the quality of the written code

Error handling

- ☐ Error conditions are routinely handled
- ☐ Assertions are used to validate logic
- ☐ The code is exception safe
- ☐ Errors are propagated, not hidden
- ☐ There are no resource leaks

- ☐ The code uses multiple threads

- ☐ It is thread safe
- ☐ There isn't potential for deadlock

Structure

- ☐ There is no redundant code
- ☐ There is no cut-and-paste programming

☐ Continue to next section ☐ Stop review here

Statement-level review

Fill out the table below, and move on to a new sheet as required. Rate issues on a scale from 0 (cosmetic/nice to have) to 5 (must fix).

File	Line	Issue	Rating
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____
_____	_____	_____	_____

Continue on a separate sheet (or mark up a paper copy of the code)

Follow-up

Record the outcome of the review here

Conclusion:

- ☐ Code OK
- ☐ Rework and verify
- ☐ Rework and re-review

Complete work by: _____

Assigned verifier: _____

Source: <https://www.codegrip.tech/>



The Ultimate Code Review Checklist

The code review process is one of those processes that differs from team to team and different standards set by developers. It is a complex process, as seen in an earlier blog and hence needs a code review checklist that every organization must follow before performing a code review.

This checklist is made for beginners as well as expert developers, stating necessary and an ideal list to do a code review process. This list is language-neutral, and you can use it for most programming languages without having to create significant changes. We made this code review checklist according to the practices that are missed by developers while building software, and hence creating poor quality code. Let's look at the comprehensive list to do a code review and build clean software.

Things to do before the code review process

There are a few points you need to take care of before performing a code review. These are practices that every team or CTO needs to do after the first draft of the code is complete.

Setting the design standard is highly essential before beginning the code review process. The team needs to lay down some measures that developers and reviewers must follow while reviewing. Further, the expectation from the software on performance, methods used, technologies implemented, and the result at the output should be noted first. This gives you a reference to check if the code is done in the required way and if not, how far did it deviate from the expectations.

Lastly, before beginning the code review process, you should always estimate the time required to do all checks in code review. The deadline and time taken to complete a code review are two leading reasons for developers ignoring it. While you don't wish to miss any step, you should always make sure that you must do checks that are more essential before those that do not contribute significantly to technical debt. Considering if you run out of time, the code would have solutions to significant problems already leaving behind some smells that would not create a bigger problem.

The Complete Code Review Checklist

Manageability

Check if the code is easily readable, easy to understand, and is highly manageable. You should do the formatting of code in such a way that it is readable. Significant steps and instructions should be commented on for better understanding, while comments that are blockers should be removed. You can delete all comments and retrieve it from an SVN file if needed. Make sure that you use proper terminology and code is aligned with appropriate spaces. Your code should be able to fit a 14-inch screen so that when imported to other monitors, it is readable.

Architecture

The code should follow an architecture throughout the whole program to be uniform. The design pattern defined earlier must be the reference when judging architecture. While reviewing if any design changes are required, be sure to document, approach, and baseline it before implementing it. The code needs to be split into different layers – presentation, business, and data layer as per requirement. Code design should resonate with earlier products and software of the same project.

Maintainability

Code review most common aim is the improvement of [code quality](#), making it **maintainable**. A good quality code has low technical debt and requires the least help in future development and manipulations. For higher code quality, make sure you maintain four factors – [code readability](#), testability, debuggability, and configurability. The code should be easy to read for any developer and must be self-explanatory. The code should be easy to test, in any way possible without failing even at edge cases. For this, try using interfaces while communicating between layers.

Correctness

This is a check for output producing the ability of code. Test plans should be present and executed, while unit cases should test all edge cases without failure. All the nonobvious logics need to be covered by tests. There should be no race around the condition.

Invalid input/states

This is a check for input taking the ability of code. Input boxes must handle all arbitrary strings as well. Check for your code's input parameters – can negatives be included?; what is the range of input?; what type of input is allowed, and if not received what case to follow? Floating-point values should have sufficient precision. If in the case of network loss, handling of the input needs to be done correctly.

Usability

Consider yourself as a user of the software that you're Developing and question yourself if the UI of the software is understandable? Any difficulty found using the software by you, who wrote the code can be a bigger problem for end-users. People rush to the development phase so early that they forget without a usable UI/API software it will result in many errors. If you are not convinced with user interface design, then start working on it with your team.

Reusability

Reusability of code is a significant factor for reducing your file length and size, saving space and also making the code much organized. See if any methods or blocks of code are not repeated in your program. Try using generic classes, functions, and components that can be reused. Follow the DRY principle (Don't Repeat Yourself) and code with no duplication.

Object-Oriented Analysis and Design (OOAD) Principles

These principles are a few checks that will make your code much more efficient. Remember all these principles are chosen according to your project, and a few may have an inverse relationship where if you follow one, the other gets void. OOAD principles are:

Single Responsibility Principle: All classes should have one responsibility, or just one function in a class or a method.

Open Closed Principle: Existing code should not be altered when new functionality is introduced.

Liskov Sustainability Principle: Having a child class should not change the meaning of the parent class.

Dependency Injection: Create dependencies outside the class and inject them to class through appropriate ways.

Interface Segregation Principle: No client should be forced to depend on methods that it does not use. Instead, create smaller interfaces based on functionality.

Source: <https://intersog.com/blog/code-review-checklist-infographic/>

Code Review Checklist Infographic

Code Review Checklist



Stop More Bugs



General



✓ Does the code work? Does it perform its intended function, the logic is correct etc.

✓ Is all the code easily understood?



✓ Does it conform to your agreed coding conventions? (cover location of braces, variable and function names, line length, indentations, formatting, and comments)

✓ Is there any redundant or duplicate code?



✓ Is the code as modular as possible?

✓ Can any global variables be replaced?



✓ Is there any commented out code?

✓ Do loops have a set length and correct termination conditions?



✓ Can any of the code be replaced with library functions?

✓ Can any logging or debugging code be removed?

Security



Are all data inputs checked (for the correct type, length, format, and range) and encoded?



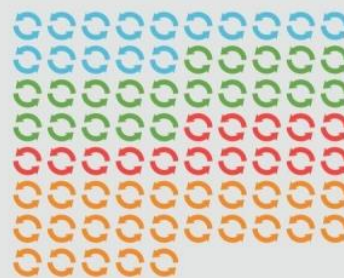
Are returning errors being caught, where third-party utilities are used?



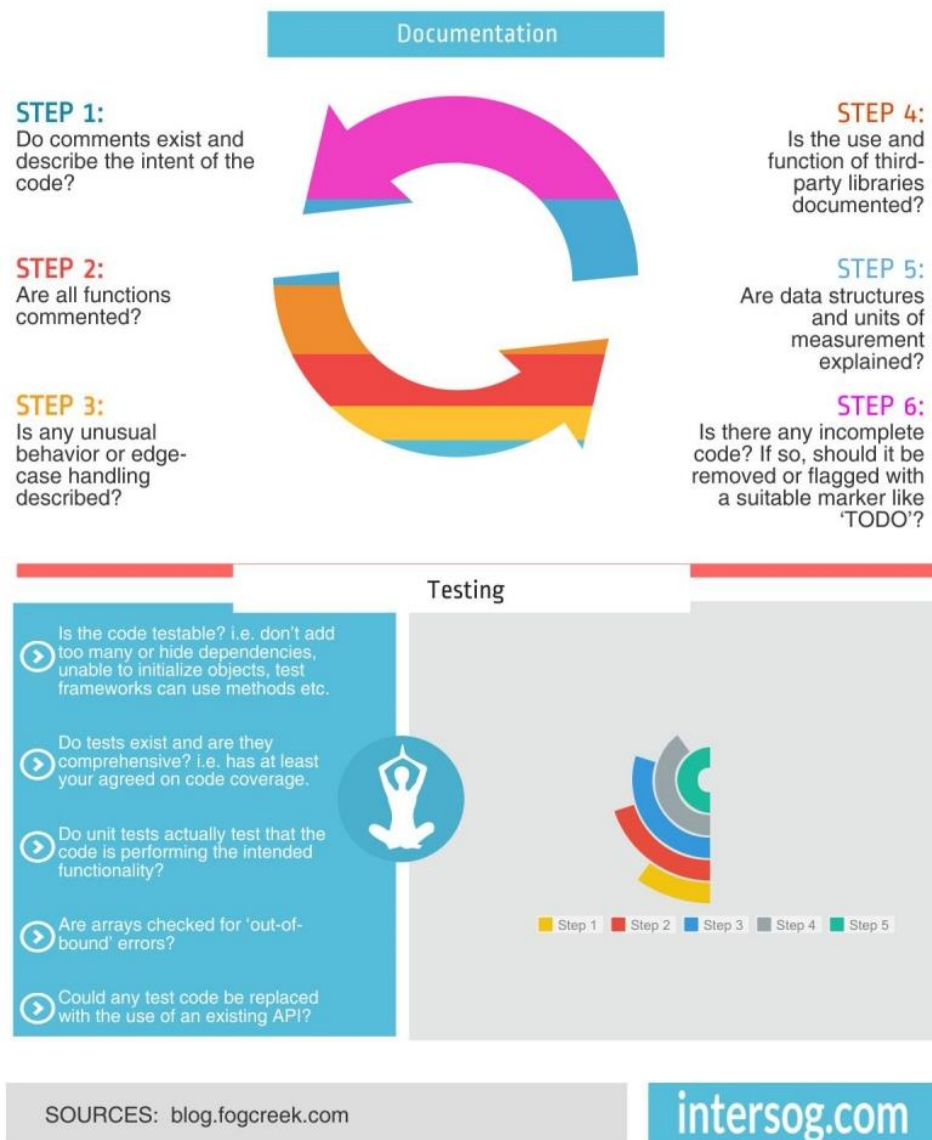
Are output values checked and encoded?



Are invalid parameter values handled?



Step 1 Step 2 Step 3 Step 4



To sum up the Checklist, when doing your code review, make sure to break it down to 4 aspects: general, security, documentation, and QA and testing.

General issues that should be checked:

- code readability and maintainability
- code compliance with pre-determined conventions
- absence of duplicates and redundancy
- length of loops, etc.

In terms of code security, make sure to check:

- all data inputs and outputs and encoding
- use of 3rd party utilities such as plug-ins and add-ons

- how invalid parameters are handled

When reviewing documentation, make sure:

- all comments have been replied to and resolved
- all unusual behaviors ever described have been addressed
- use of 3rd party libraries is well documented
- your data structures and units are explained

Regarding testing, make sure your code is testable and all tests are comprehensive enough, all arrays are checked for out-of-bound errors, etc.

What other issues would you add to this Checklist?

Source: <https://blog.fogcreek.com/>

What is code review?

Code review happens when another developer goes through you or your team's code line-by-line and provides constructive, helpful feedback. Code review saves time and effort by ensuring code quality up-front, rather than waiting until issues are discovered in production. It's an essential part of the day-to-day lives of many professional software developers and data scientists.

Why Code Reviews are Important

The main purpose of code review is to prevent problematic code from being deployed to production. Reviews can help you catch bugs, identify missed edge cases, spot design issues, and avoid anti-patterns before they become a problem.

The second purpose of code review is to help you be a better developer. When you know that your code is going to be reviewed by another developer, you write code differently. You take the time to give methods or functions names that express what they do. You add more thorough tests. You write code that's meant to be readable. You spend more time thinking through side effects and unintended consequences.

Code Review Checklist

If you are an experienced developer who can go through someone else's code line by line to help check their approach to a problem, here is a list of essential questions to ask yourself during this process:

- Are there potential problems with the developer's approach? Does the approach make sense?
- Are there any security vulnerabilities written in the code?
- Is there a way to do this faster or better?
- Would you want to be responsible for changes to this code? What would you do differently?
- Does this code follow the best practices for this language/framework/library?
- Can the code be easily maintained?

Once your comments are all set, go over them again and check:

- Are all of your comments critical? Or did you also provide positive feedback?
- Are your comments specific and clear? Did you give concrete examples whenever possible?
- Are you using any words or acronyms that the developer may not understand?

Is there anything else that the developer might have missed?

Source: <https://www.evoketechnologies.com/blog/>

10 Simple Code Review Tips for Effective Code Reviews

Software code review is a process to ensure that the code meets the functional requirements and also helps the developers to adhere to the best coding practices. Additionally, code review process helps in improving the software quality.

Based on my experience, would like to share 10 simple code review tips, which would help code reviewers and software developers during their code reviews.

1. Highlight issues in the code

Never force software developers to change the code written by them. It may hurt their ego and they may repeat the same mistake if they do not understand the reason behind code change recommendation. Highlight the issues in the existing code and its consequences.

Here's an interesting quote on this point: *"If an egg is broken by outside force, life ends. If broken by inside force, life begins. Great things always begin from inside."* – Jim Kwik, Learning Expert.

2. Explain relevant principles

If software developers hesitate to accept given suggestions/recommendations, then explain them relevant principles such as [Separation of Concerns](#), [SRS](#) (Single Responsibility Principle), [Open-Closed principle](#), [Cyclomatic complexity](#). If necessary, discuss with them the Non Functional Requirements (NFR) such as Maintainability, Extensibility, Testability and Reliability.

3. Discuss relevant quotes

To make the code review process more interesting and engrossing, remind developers relevant quotes/proverbs:

- *"Any stupid can write the program that computer understands but only good programmers write code that humans understand"* – [Martin Fowler](#)
- *"Measuring programming progress by lines of code is like measuring aircraft building progress by weight."* – Bill Gates
- *"Fat model and thin controller", "High cohesion and Low coupling"*
- *"When debugging, novices insert corrective code; experts remove defective code."* – Richard Pattis

4. Do few things offline

Instead of explaining the entire solution to developers during the code review process, simply share the links of relevant websites or encourage them to research on the internet by providing keywords. This action would certainly save the code reviewer's time and energy. And of course, developers would also like it, since they too need some time to assimilate the proposed solution.

Instead of always sitting next to a developer during the code review process, code reviewers should obtain the code from the source control or shared path, so that it saves developers time. And this would also give code reviewers ample time to recommend the best solution in the context.

5. Consider as an Opportunity to learn best practices

Sometimes software developers may take the code reviewer's comments personally and defend the code without a valid reason. It then becomes the responsibility of a code reviewer to inform the developer to consider this exercise as an opportunity to learn/discuss best practices, but not to identify issues to criticize. Ideally, code reviewers should inform the managers that code review comments should not be used to assess a software developer's skills. Code review should always be done in a competitive spirit to find more useful comments.

6. Always be patient and relook if required

Sometimes, developers do not accept suggestions/recommendation and keep debating. A code reviewer may not know the exact context and challenges, when the code was written. A code reviewer should understand all the points being made by the developer without losing patience. Furthermore, to make the point crystal clear, a code reviewer can explain the points on a paper or on a whiteboard by comparing the developer's approach vs code reviewer's approach. Every approach has its pros and cons, need to choose the right approach, whichever weighs more after careful evaluation.

Many times, a third approach evolves which is acceptable to both the developer and the code reviewer. If both of them do not come to a conclusion, then stop the discussion by saying "Let's discuss this tomorrow, after doing some more analysis". If the same issue is re-looked on another day with a fresh mindset, it is quite likely that a new approach evolves. Always remember that *"No problem can be solved from the same level of consciousness that created it."* – [Albert Einstein](#)

7. Explain the need for best coding practices

Generally, software developers mention that best coding practices are not followed due to tight project schedules. Developers may feel that it is an acceptable practice. However, code reviewers should educate software developers that as the code size increases or after sometime, the application

becomes very difficult to maintain. Moreover, if a client verifies the code then poor quality code may give wrong impression on the team's/organization's quality standards. It may also impact awarding new projects or referring an organization to prospective clients.

If the project schedules are too tight then code reviewers should suggest developers to perform [code refactoring](#) while fixing a defect/adding an enhancement or in next version. While refactoring the code, some functionality may break accidentally. Code reviewers should convince the project managers by explaining the importance of code refactoring and the need for allocating additional time to plan this activity.

8. Consult second level code reviewer (if not convinced)

If a code reviewer recommends few suggestions, but the developer hesitates to accept these, then discuss it out with the developer. It is quite possible that the code reviewer may not know the entire context. If the developer is still not convinced with the recommendations of the code reviewer, it is perfectly all right to consult a second level code reviewer. However, the developer should ensure that second code reviewer's suggestions are forwarded to the first code reviewer to ensure that everyone is on the same page.

9. Capture the enhancements and technical debt

It is quite likely that some code review suggestions cannot be implemented during current release. However, a code reviewer should ensure that all accepted recommendations are clearly documented in a shared code review document, so that these are implemented at an appropriate time in future. Additionally, code reviewer should identify and capture all the enhancements from technology and business perspective. Once the project is completed, all captured enhancements can be considered for implementation, instead of searching at that moment. Finding enhancements during code reviews is more efficient than finding separately at the end of the project.

10. Document all code review comments

Document all code review comments in an email, word document, excel, or any standard tool used by the organization. Making a mistake for the first time is acceptable, but it is not a good sign to repeat the same mistake. The code review document helps software developers to cross check the highlighted

issues and avoid making similar mistakes in future. Additionally, maintaining a code review document is a mandatory part of the Capability Maturity Model Integration (CMMI) level process.

Conclusion

The above code review tips would help code reviewers and developers to perform effective code reviews. The code review process should always be pursued in a constructive way by all stakeholders to gain maximum benefit.

The code review process not only improves the [software quality](#) but also helps software developers to enhance their skills continuously. Hence, organizations/project managers must ensure that code reviews are made integral part of software development lifecycle.

Code Review Checklist – To Perform Effective Code Reviews

Code Review Checklist	
<input checked="" type="checkbox"/>	<u>Coding standards</u>
<input type="checkbox"/>	<u>Coding Best practices</u>
<input checked="" type="checkbox"/>	<u>Non Functional Requirements</u>
<input checked="" type="checkbox"/>	<u>OOAD Principles</u>
<input checked="" type="checkbox"/>	<u>Static Code Analysis Metrics</u>
<input type="checkbox"/>	<u>*****</u>

- [Code Reviews at Google are lightweight and fast](#)
- [A Code Review Checklist – Focus on the Important Issues](#)

Code Review Checklist

Implementation

- Does this code change do what it is supposed to do?
- Can this solution be simplified?
- Does this change add unwanted compile-time or run-time dependencies?
- Was a framework, API, library, service used that should not be used?
- Was a framework, API, library, service not used that could improve the solution?
- Is the code at the right abstraction level?
- Is the code modular enough?
- Would you have solved the problem in a different way that is substantially better in terms of the code's maintainability, readability, performance, security?
- Does similar functionality already exist in the codebase? If so, why isn't this functionality reused?
- Are there any best practices, design patterns or language-specific patterns that could substantially improve this code?
- Does this code follow Object-Oriented Analysis and Design Principles, like the Single Responsibility Principle, Open-close Principle, Liskov Substitution Principle, Interface Segregation, Dependency Injection?

Logic Errors and Bugs

- Can you think of any use case in which the code does not behave as intended?
- Can you think of any inputs or external events that could break the code?

Error Handling and Logging

- Is error handling done the correct way?
- Should any logging or debugging information be added or removed?
- Are error messages user-friendly?
- Are there enough log events and are they written in a way that allows for easy debugging?

Usability and Accessibility

- Is the proposed solution well designed from a usability perspective?
- Is the API well documented?
- Is the proposed solution (UI) accessible?
- Is the API/UI intuitive to use?

Testing and Testability

- Is the code testable?
- Does it have enough automated tests (unit/integration/system tests)?

Dependencies

- If this change requires updates outside of the code, like updating the documentation, configuration, readme files, was this done?
- Might this change have any ramifications for other parts of the system, backward compatibility?

Security and Data Privacy

- Does this code open the software for security vulnerabilities?
- Are authorization and authentication handled in the right way?
- Is sensitive data like user data, credit card information securely handled and stored?
- Is the right encryption used?
- Does this code change reveal some secret information like keys, passwords, or usernames?
- If code deals with user input, does it address security vulnerabilities such as cross-site scripting, SQL injection, does it do input sanitization and validation? Is data retrieved from external APIs or libraries checked accordingly?

Readability

- Was the code easy to understand?
- Which parts were confusing to you and why?
- Can the readability of the code be improved by smaller methods?
- Can the readability of the code be improved by different function/method or variable names?
- Is the code located in the right file/folder/package?
- Do you think certain methods should be restructured to have a more intuitive control flow?
- Is the data flow understandable?
- Are there redundant comments?
- Could some comments convey the message better?
- Would more comments make the code more understandable?
- Could some comments be removed by making the code itself more readable?
- Is there any commented out code?

Experts Opinion

- Do you think a specific expert, like a security expert or a usability expert, should look over the code before it can be committed?

- [Generic Checklist for Code Reviews](#)
